

# The ITOS Events API

---

Integrated Test & Operations System

5 October 2006

---

Copyright 1999-2006, United States Government as represented by the Administrator of the National Aeronautics and Space Administration. No copyright is claimed in the United States under Title 17, U.S. Code.

This software and documentation are controlled exports and may only be released to U.S. Citizens and appropriate Permanent Residents in the United States. If you have any questions with respect to this constraint contact the GSFC center export administrator, <Thomas.R.Weisz@nasa.gov>.

This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD. See <<http://itos.gsfc.nasa.gov/>> or e-mail <[itos@itos.gsfc.nasa.gov](mailto:itos@itos.gsfc.nasa.gov)> for additional information.

You may use this software for any purpose provided you agree to the following terms and conditions:

1. Redistributions of source code must retain the above copyright notice and this list of conditions.
2. Redistributions in binary form must reproduce the above copyright notice and this list of conditions in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:  
This product contains software from the Integrated Test and Operations System (ITOS), a satellite ground data system developed at the Goddard Space Flight Center in Greenbelt MD.

This software is provided ‘‘as is’’ without any warranty of any kind, either express, implied, or statutory, including, but not limited to, any warranty that the software will conform to specification, any implied warranties of merchantability, fitness for a particular purpose, and freedom from infringement and any warranty that the documentation will conform to their program or will be error free.

In no event shall NASA be liable for any damages, including, but not limited to, direct, indirect, special or consequential damages, arising out of, resulting from, or in any way connected with this software, whether or not based upon warranty, contract, tort, or otherwise, whether or not injury was sustained by persons or property or otherwise, and whether or not loss was sustained from or arose out of the results of, or use of, their software or services provided hereunder.

## Introduction

ITOS includes an events subsystem which is responsible for collecting event messages from all processes in ITOS, logging them, displaying them, and distributing them. Applications within ITOS employ a simple API to send messages to the events subsystem, and this document describes that API and its operation.

The API is contained in the main ITOS library, `'libtcw.so'`. (A library containing only the Event API, called `'libitosevt.so'`, also is available .)

## 1 Opening the Event Queue

To open a connection to the ITOS events subsystem, programs call the function `open_event()`. This function should be called before sending messages to the events subsystem.

This function opens a FIFO (also known as a *named pipe*) given by the environment variable `ITOS_EVTFIFO`. If `ITOS_EVTFIFO` is not set, the function tries to open `./fifos/event_fifo`. Upon opening the FIFO, it closes standard error, duplicates the FIFO file descriptor making the FIFO standard error, and closes the original FIFO file descriptor.

The `open_event()` function returns a 0 upon successful completion. On errors, it returns -1 and sets the global `char *liberrmsg` to a string indicating the error that occurred.

Note that if `open_event()` fails, applications can still call `EvtMsg()`, but the messages will go to the standard error output.

## 2 Sending Event Messages

To send event messages, ITOS programs call the function `EvtMsg()`, which has the following signature:

```
void EvtMsg(enum event_types event_type, char *format, ...)
```

and is used like the standard C function `printf()`. The *format* is a `printf()`-style format, with the extensions available to the ITOS function `Snprintf()`. The *event\_type* is an enumerated constant giving the type of event being generated. The ITOS event windows allow user to filter messages based on these event types.

Details on the format extensions and event types are provided in the following sections.

### 2.1 How to use `EvtMsg()` and `EvtMsgPurge()`

The `EvtMsg()` function queues text within the calling application for output to the ITOS event FIFO. Text is actually written to the event FIFO when a newline character is queued, or when the *event\_type* changes. In the latter case, the string `['\n???']` is appended at the end of the event message.

Some examples will help explain this behavior. The statements

```
EvtMsg(CMD_MSG, "This is a CMD_MSG event\n");
```

and

```
EvtMsg(CMD_MSG, "This is a ");
EvtMsg(CMD_MSG, "CMD_MSG event\n");
```

produce identical event messages. The statement

```
EvtMsg(IN_LIMITS, "One\nTwo\n");
```

produces two `IN_LIMITS` event messages. The statements

```
EvtMsg(RED_VIOL, "This is a ");
EvtMsg(YEL_VIOL, "YELLOW violation\n");
```

produces two event messages, the `RED_VIOL` message `'This is a [\n???']` and the `YEL_VIOL` message `'YELLOW violation'`. Finally,

```
EvtMsg(TM_MSG, "%d + %d = 0x%08x\n", 15, 1, 16);
```

produces the `TM_MSG` message `'15 + 1 = 0x00000010'`.

An additional function, `EvtMsgPurge()`, clears any event message text queued in the local application. This might be used in cases where it is convenient to create the beginning of an error message just in case an error occurs. `EvtMsgPurge()` provides a way to discard that message if no error actually occurs.

### 2.2 What `EvtMsg()` writes

When `EvtMsg()` writes a message, it writes a two-digit ASCII integer corresponding to the *event\_type* followed immediately, without space, by exactly what `printf()` (or, really, `Snprintf()`) would write.

From the examples in the previous section, then, the statement

```

    EvtMsg(CMD_MSG,"This is a CMD_MSG event\n");
writes
    11This is a CMD_MSG event\n

```

The '11' at the head of the output string is the event code. The numbers corresponding to the `enum event_types` constants can be found in Section 2.4 [Event Types], page 4.

## 2.3 Format Extensions

The following format extensions are available to users of `EvtMsg()`:

- `%m` is replaced with the error message corresponding to the current value of the global error code `errno`, as it would be returned from the function `strerror()`. There must be no argument in the argument list corresponding to this conversion.
- `%b` is used to print integer or unsigned values in binary representation.

## 2.4 Event Types

The `event_type` argument to `EvtMsg()` must be an enumerated constant from the following list. The interpretation of each event type is left to the programmer.

<code>enum event_type</code>	code	meaning
<code>NULL_EVENT</code>	00	Unknown event type; can't be filtered.
<code>RED_VIOL</code>	01	A red limits violation occurred.
<code>YEL_VIOL</code>	02	A Yellow limits violation occurred.
<code>DEL_VIOL</code>	03	A Delta limits violation occurred.
<code>IN_LIMITS</code>	04	A value went back in limits.
<code>TM_MSG</code>	05	Telemetry informational message.
<code>TM_WARN</code>	06	Telemetry warning message.
<code>TM_ERROR</code>	07	General telemetry error message.
<code>CMD_EVENT</code>	08	Command event.
<code>CMD_VERIFY</code>	09	Command verify/no-verify message.
<code>CFG_ERROR</code>	10	Configuration error message.
<code>CMD_MSG</code>	11	Command informational message.
<code>CMD_WARN</code>	12	Command warning message.
<code>CMD_ERROR</code>	13	General command error message.
<code>CMD_TF</code>	14	Command transfer frame echoed in hex.
<code>OPER_ERROR</code>	15	STOL Operator error.
<code>STOL_ECHO</code>	16	STOL echo of directives.
<code>STOL_MSG</code>	17	STOL MSG directive message.
<code>STOL_WARN</code>	18	STOL warning message.
<code>STOL_ERROR</code>	19	STOL error message.
<code>DSP_MSG</code>	20	Display informational message.
<code>DSP_WARN</code>	21	Display warning message.
<code>DSP_ERROR</code>	22	Display error message.
<code>STOL_PRE</code>	23	STOL directive preview message.

CMD_DECODE	24	Command decoded back from the spacecraft.
unused	25	unused.
SYS_ERROR	26	System call failure message – call a programmer!.
TCW_FAULT	27	Serious error message – call a programmer!.
SC_EVENT	28	Spacecraft event message.
CFG_ALERT	29	Configuration monitor alert message.
DEBUG_EVT	30	A debugging message.
SDP_MSG	31	Science Data Processing message.
SDP_WARN	32	Science Data Processing warning.
SDP_ERROR	33	Science Data Processing error.
CTLR_MSG	34	Controller message.
CTLR_WARN	35	Controller warning.
CTLR_ERROR	36	Controller error.
CFDP_MSG	37	CFDP Driver message.
CFDP_WARN	38	CFDP Driver warning.
FDP_ERROR	39	CFDP Driver error.

# Index

## %

`%b` ..... 4  
`%m` ..... 4

## C

clearing queued messages ..... 3

## E

`enum event_types` ..... 3, 4  
event type ..... 3, 4  
`EvtMsg()` ..... 3, 4  
`EvtMsgPurge()` ..... 3

## F

FIFO ..... 2  
format ..... 3, 4  
format extensions ..... 4

## L

`libtcw.so` ..... 1

## N

named pipe ..... 2  
newline character ..... 3

## O

`open` ..... 2  
`open_event()` ..... 2

## P

`printf()` ..... 3

## Q

queue ..... 3

## S

`Sprintf()` ..... 3  
standard error ..... 2



## Table of Contents

<b>Introduction .....</b>	<b>1</b>
<b>1 Opening the Event Queue .....</b>	<b>2</b>
<b>2 Sending Event Messages .....</b>	<b>3</b>
2.1 How to use EvtMsg() and EvtMsgPurge() .....	3
2.2 What EvtMsg() writes .....	3
2.3 Format Extensions .....	4
2.4 Event Types .....	4
<b>Index .....</b>	<b>6</b>